Software Life-cycle and Integration Issues for CS&E R&D Software and Experiences from Trilinos (Part I)

Roscoe A. Bartlett
http://www.cs.sandia.gov/~rabartl/
Department of Optimization & Uncertainty Estimation
Trilinos Software Engineering Technologies and Integration Lead

Sandia National Laboratories

SIAM Parallel Processing 2010, 2/24/2010

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Sandia National Laboratories

Overview of CS&E Software Engineering Challenge

- Progress in Computational Science and Engineering (CS&E) is occurring due to greater numbers of more complex algorithms and methods
  - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, …
  - **Parallelization**: a) parallel support, b) load balancing, …
  - **General numerics**: a) automatic differentiation, …
  - **Solvers**: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers, e) time integration, …
  - **Analysis capabilities**: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
  - **New architectures**: a) multi-core, b) GPUs, …
  - **Visualization**
  - ...
- Each technology requires specialized PhD-level expertise
- Almost all technologies need to be integrated into single applications
- Set of algorithms/software is too large for any single organization to create
- Too large to be developed under single blanket of Continuous Integration (CI)

Software Engineering and Software Integration are key bottlenecks for CS&E to have the fullest impact!

Overview of Trilinos

Sandia National Laboratories

Evolving Trilinos Solution

- **Trilinos[1] is an evolving framework to address these challenges:**
  - **Follow a TOOLKIT approach.**
  - **Fundamental atomic unit is a *package*.**
  - **Includes core set of vector, graph and matrix classes (Epetra/Tpetra packages).**
  - **Provides a common abstract solver API (Thyra package).**
  - **Provides a ready-made package infrastructure:**
    - **Source code management (git [New]).**
    - **Build tools (CMake [New]).**
    - **Automated regression testing (CTest/CDash [New]).**
    - **Communication tools (Mailman mail lists).**
  - **Specifies requirements and suggested practices for package SQA.**
- **In general allows us to categorize efforts:**
  - **Efforts best done at the Trilinos framework level (useful to most or all packages).**
  - **Efforts best done at a package level (peculiar or important to a package).**
  - Allows package developers to focus only on things that are unique to their package.

1. Trilinos loose translation: “A string of pearls”

Trilinos Package Summary

	Objective	Package(s)
Discretizations	Meshing & Spatial Discretizations	phdMesh, Intrepid, Phalanx, Shards, Pamgen, Sundance, FEI, STK, Mesquite
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Core	Linear algebra objects	Epetra, Jpetra, Tpetra
	Abstract interfaces	Thyra, Stratimikos, RTOp
	Load Balancing	Zoltan, Isorropia
	“Skins”	PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos
	C++ utilities, I/O, thread API	Teuchos, EpetraExt, Kokkos, Triutils, TPI, Optika
Solvers	Iterative (Krylov) linear solvers	AztecOO, Belos, Komplex, Lyno
	Direct sparse linear solvers	Amesos
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi
	ILU-type preconditioners	AztecOO, IFPACK, Tifpack
	Multilevel preconditioners	ML, CLAPS
	Block preconditioners	Meros, Teko
	Nonlinear system solvers	NOX, LOCA, Stalix
	Optimization	MOOCHO, Aristos, GlobiPack, OptiPack, TriKota, Piro
	Stochastic PDEs	Stokhos, Stalix

Number of Trilinos Packages:

- Current = 57
- Growth = 5-10 new packages per year!

Number of Trilinos External Third Partly Libraries (TPLs) = 42

Sandia National Laboratories

Trilinos Strategic Goals

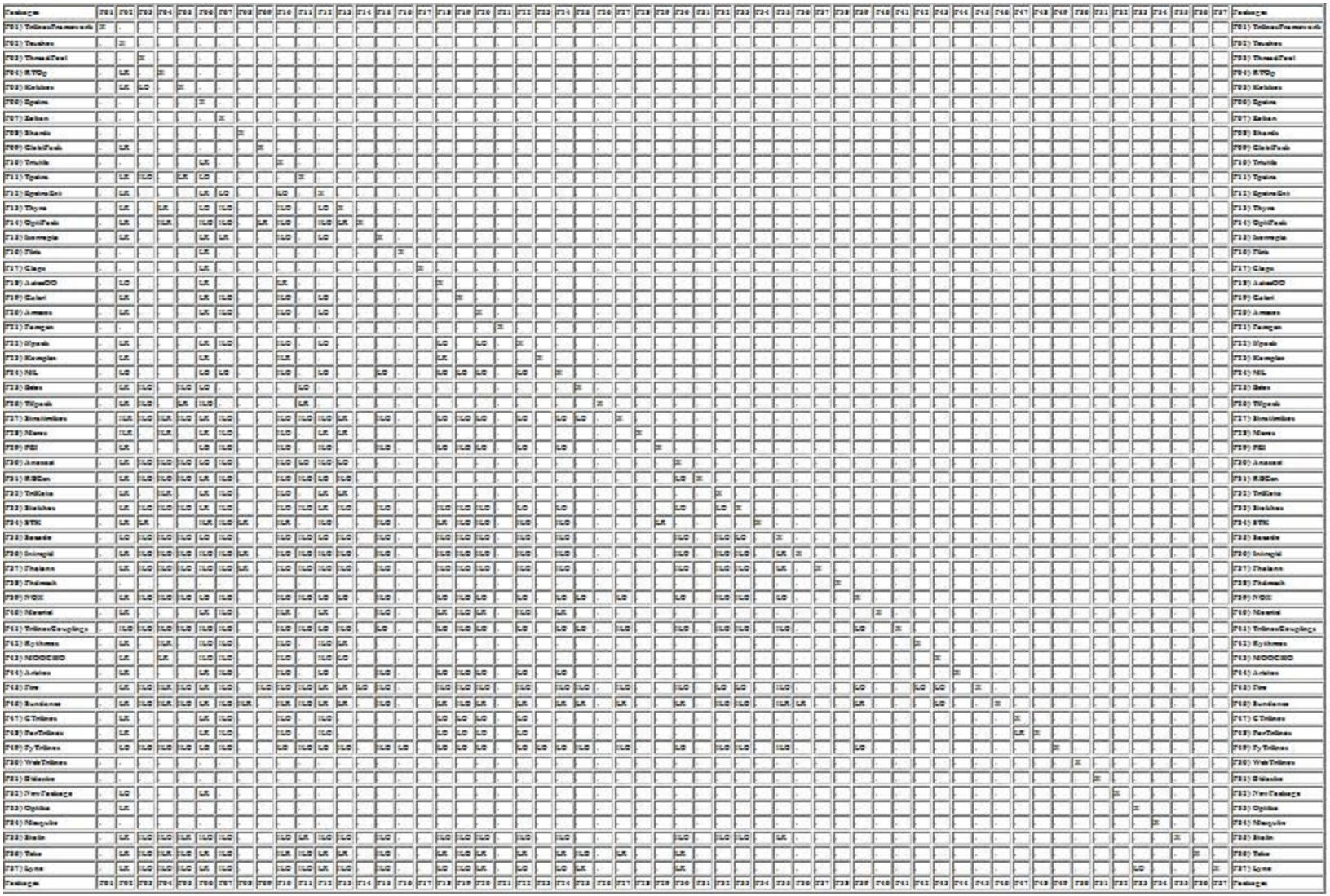
- **Scalable Computations:** As problem size and processor counts increase, the cost of the computation will remain nearly fixed.
- **Hardened Computations:** Never fail unless problem essentially intractable, in which case we diagnose and inform the user why the problem fails and provide a reliable measure of error.
- **Full Vertical Coverage:** Provide leading edge enabling technologies through the entire technical application software stack: from problem construction, solution, analysis and optimization.

Algorithmic Goals

- ***Grand* Universal Interoperability**: All Trilinos packages will be interoperable, so that any combination of solver packages that makes sense algorithmically will be possible within Trilinos.
- **Universal Accessibility:** All Trilinos capabilities will be available to users of major computing environments: C++, Fortran, Python and the Web, and from the desktop to the latest scalable systems.
- **Universal Solver RAS**: Trilinos will be:
  - Reliable: Leading edge hardened, scalable solutions for each of these applications
  - Available: Integrated into every major application at Sandia
  - Serviceable: Easy to maintain and upgrade within the application environment.

Software Goals

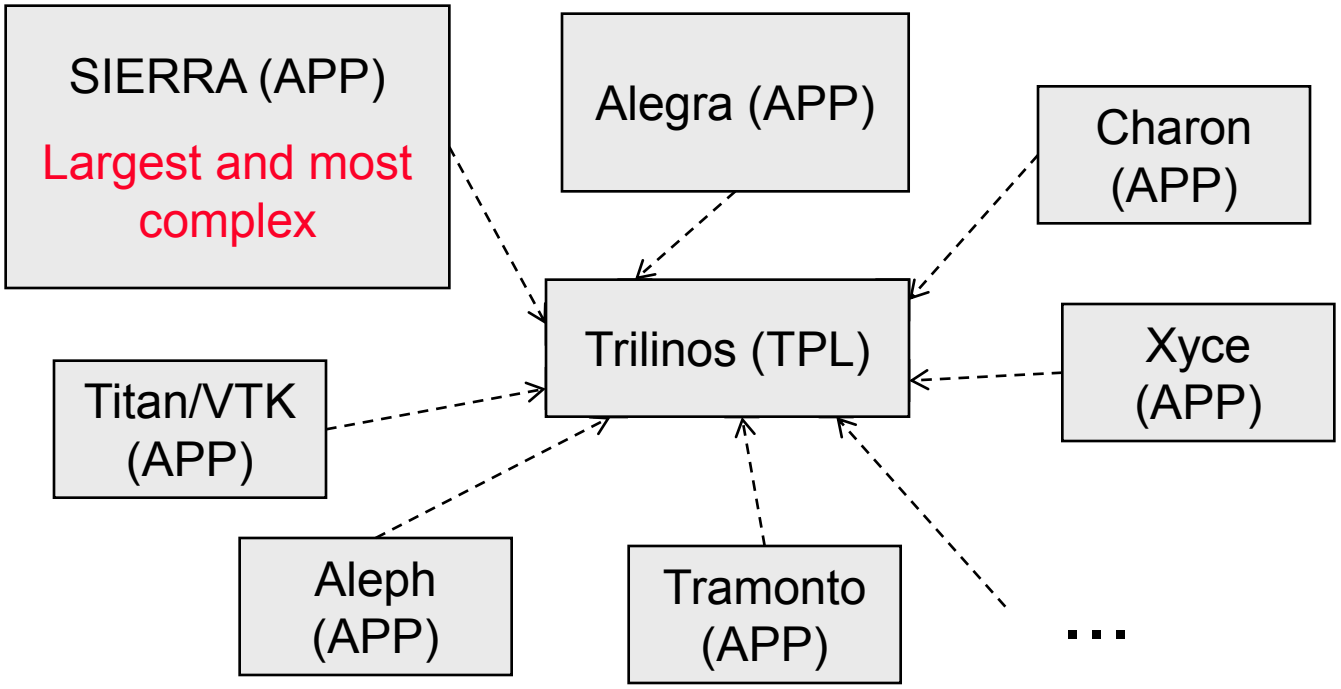
Automatic Intra-Package Dependency Handling



http://trilinos.sandia.gov/packages/interoperability.html

Sandia National Laboratories

CS&E Environment at Sandia National Labs for Trilinos



TPL: Third Party Lib

- Provides functionality to multiple APPs
- The "Supplier" to the APP

APP: Application

- Delivers end user functionality
- The "Customer" of the TPL

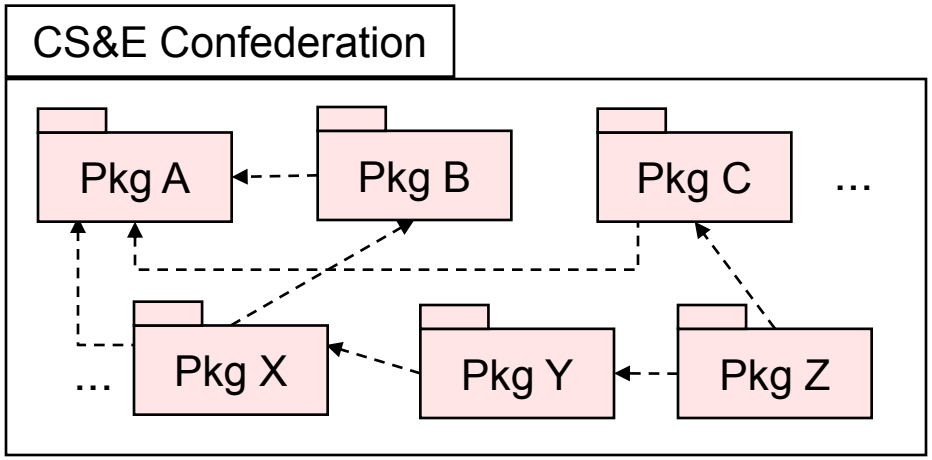
- Sophisticated CS&E applications
  - Discretized PDEs (SIERRA, Alegra, Aleph, Charon)
  - Circuit network models (Xyce)
  - Other types of calculations (Titian/VTK, Tramonto)
- (Massively) parallel MPI (Gordon Bell Winners)
- Almost entirely developed by non-software people
- Wide range of research to production (i.e. from Aleph to SIERRA)

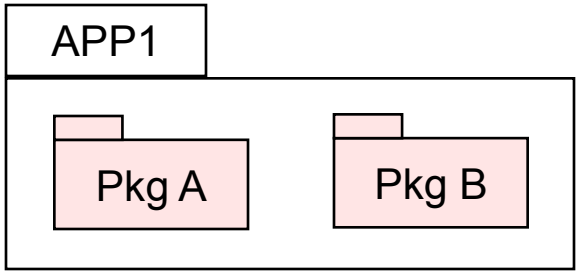
Vision for a Confederation of CS&E Software?

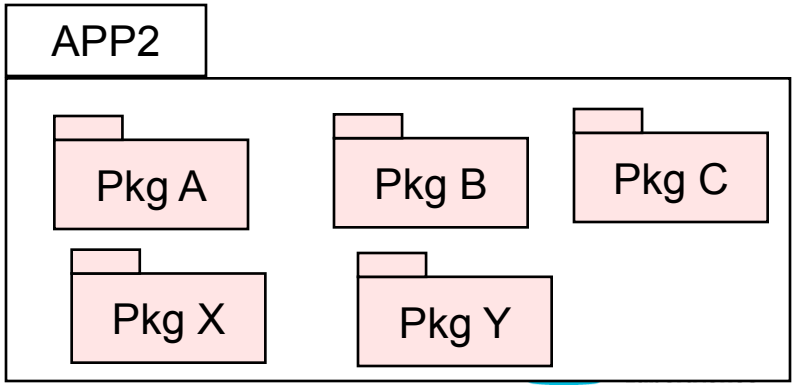
Sandia National Laboratories

The Vision: A Confederation of CS&E Codes?

- Develop a confederation of trusted, high-quality, reusable, compatible, software packages/components including capabilities for:
  - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, …
  - **Parallelization**: a) parallel support, b) load balancing, …
  - **General numerics**: a) automatic differentiation, …
  - **Solvers**: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers, e) time integration, …
  - **Analysis capabilities**: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
  - **New architectures**: a) multi-core, b) GPUs, …
  - **Visualization**
  - …







Trilinos itself is a smaller example of this!

Challenges for the Reuse of External CSE Software

Many CS&E organizations and individuals are adverse to using externally developed CS&E software!

Using externally developed software can be as risk!

- External software can be hard to learn
- External software may not do what you need
- Upgrades of external software can be risky:
  - Breaks in backward compatibility?
  - Regressions in capability?
- External software may not be well supported
- External software may not be support over long term

What can reduce the risk of depending on external software?

- Develop and maintain strong organizational relationships
- Long term commitment and support (i.e. 10-30 years)
- Apply strong software engineering processes and practices (high quality, low defects, frequent releases, regulated backward compatibility)

Overview of Modern Lean/Agile Software Engineering

Sandia National Laboratories

Defined: Life-Cycle, Agile and Lean

- **Software Life-Cycle:** The processes and practices used to design, develop, deliver and ultimately discontinue a software product or suite of software products.
  - Example life-cycle models: Waterfall, Spiral, Evolutionally Prototype, Agile, …
- **Agile Software Engineering Methods:**
  - Agile Manifesto (2001) (Capital 'A' in Agile)
  - Founded on long standing wisdom in SE community (40+ years)
  - Push back against heavy plan-driven methods (CMM(I))
  - Focus on incremental design, development, and delivery (i.e. software life-cycle)
  - Close customer focus and interaction and constant feedback
  - Example methods: SCRUM, XP (extreme programming)
  - Becoming a dominate software engineering approach
- **Lean Software Engineering Methods**:
  - Adapted from Lean manufacturing approaches (e.g. the Toyota Production System).
  - Focus on optimizing the value chain, small batch sizes, minimize cycle time, automate repetitive tasks, …
  - Hard to distinguish between Lean and Agile …

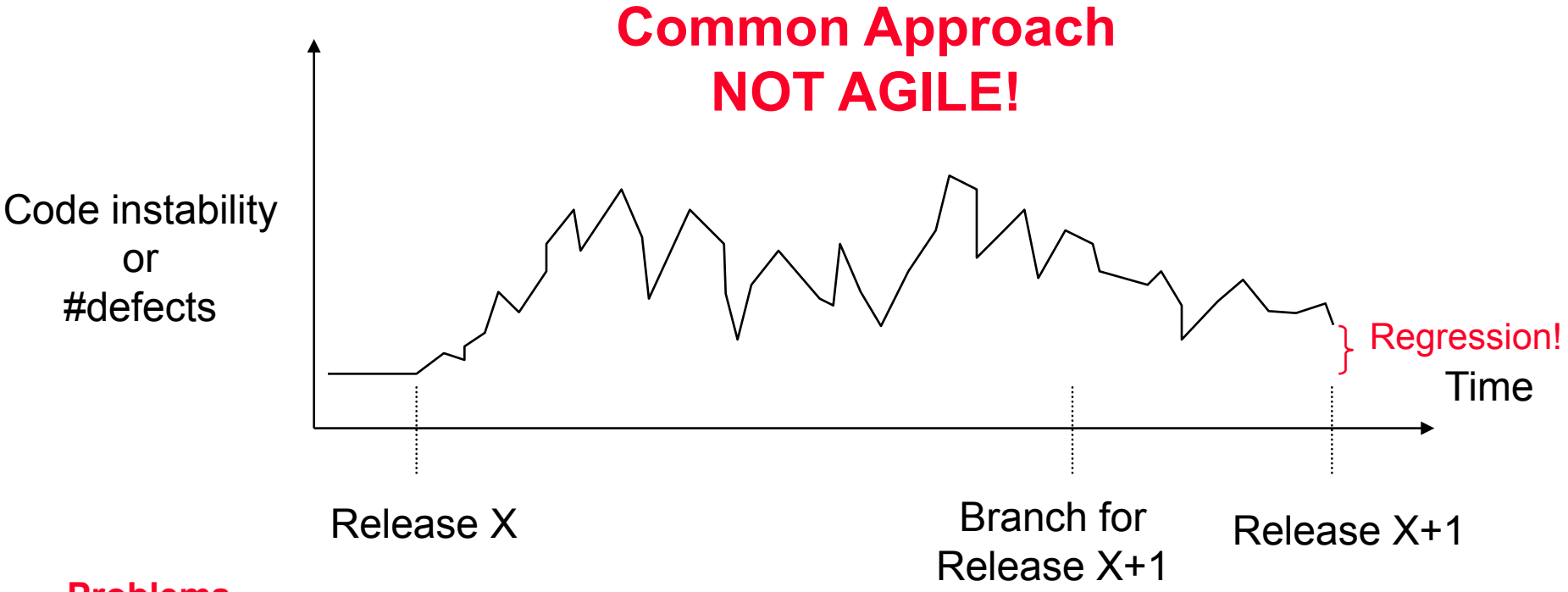
References: http://www.cs.sandia.gov/~rabartl/readingList.html

Relevant Lean/Agile Software Engineering Principles

- **Agile Design:** Reusable software is best designed and developed by incrementally attempting to reuse with new clients and incrementally redesigning and refactoring the software as needed keeping it simple.
  - Technical debt in the code is management through continuous incremental (re)design and refactoring.
- **Agile Quality**: High quality defect-free software is most effectively developed by not putting defects into the software in the first place.
  - High quality software is best developed collaboratively (e.g. pair programming and code reviews).
  - Software is fully verified before it is even written (i.e. Test Driven Development for system verification and unit tests).
  - High quality software is developed in small increments and with sufficient testing in between sets of changes.
- **Agile Integration**: Software needs to be integrated early and often
- **Agile Delivery**: Software should be delivered to real (or as real as we can make them) customers is short (fixed) intervals.

References: http://www.cs.sandia.gov/~rabartl/readingList.html

Lean/Agile Methods: Development Stability

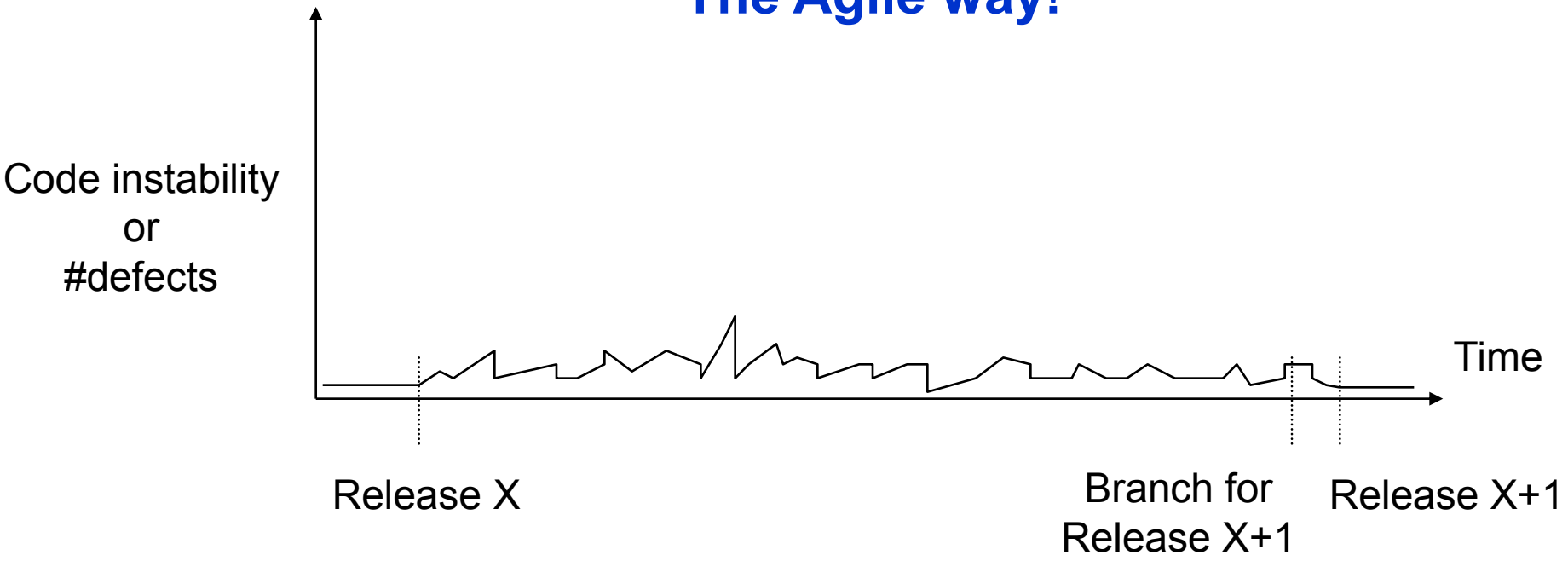


Problems

- Cost of fixing defects increases the longer they exist in the code
- Difficult to sustain development productivity
- Broken code begets broken code (i.e. broken window phenomenon)
- Long time between branch and release
  - Difficult to merge changes back into main development branch
  - Temptation to add “features” to the release branch before a release
- High risk of creating a regression

Lean/Agile Methods: Development Stability

The Agile way!



Advantages

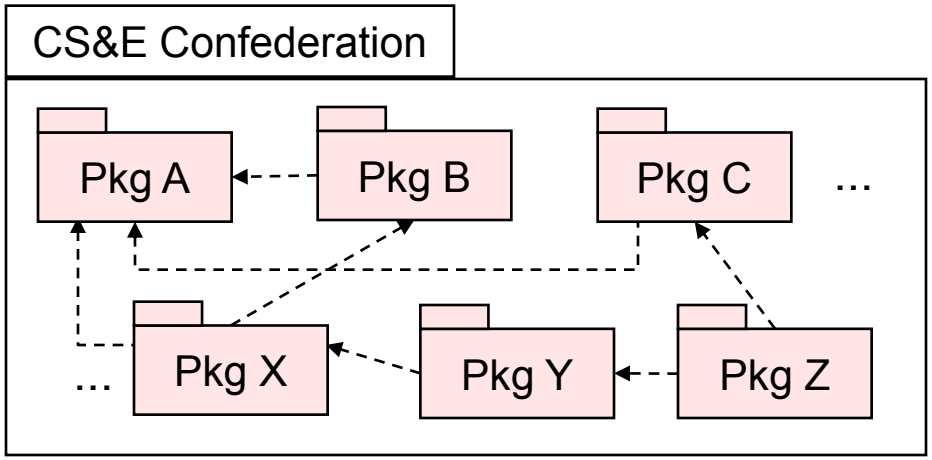
- Defects are kept out of the code in the first place
- Code is kept in a near releasable state at all times
- Shorten time needed to put out a release
- Allow for more frequent releases
- Reduce risk of creating regressions
- Decrease overall development cost

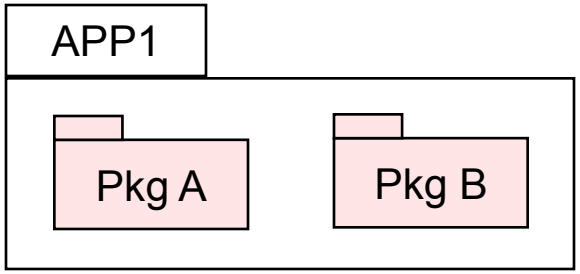
Realizing a Confederation of CS&E Software

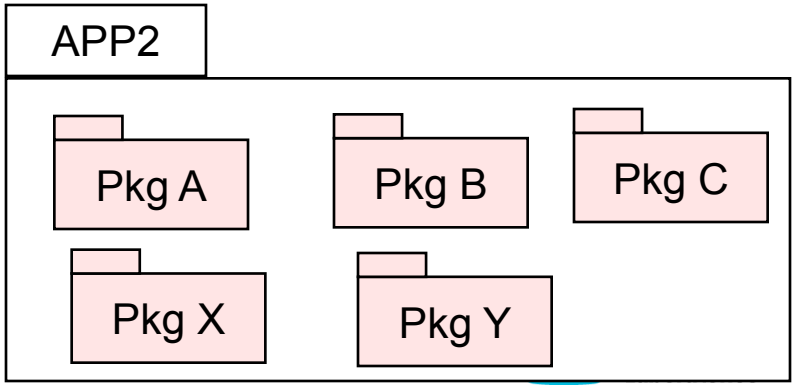
Sandia National Laboratories

The Vision: A Confederation of CS&E Codes?

- Develop a confederation of trusted, high-quality, reusable, compatible, software packages/components including capabilities for:
  - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, …
  - **Parallelization**: a) parallel support, b) load balancing, …
  - **General numerics**: a) automatic differentiation, …
  - **Solvers**: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers, e) time integration, …
  - **Analysis capabilities**: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
  - **New architectures**: a) multi-core, b) GPUs, …
  - **Visualization**
  - …







Trilinos itself is a smaller example of this!

Requirements/Challenges for Confederation of CS&E Codes

- Software quality and usability
  - => Design, testing, collaborative development
- Building the software in a consistent way and linking
  - => Common build approach?
- Reusability and interoperability of software components
  - => Incremental Agile design, resource management, …
- Documentation, tutorials, user comprehension
  - => SE education, better documentation and examples
- Critical new functionality development
  - => Closer development and integration models
- Upgrading compatible versions of software
  - => Frequent fixed-time releases
- Safe upgrades of software
  - => Regulated backward compatibility, software quality
- Long term maintenance and support
  - = > Stable organizations, stable projects, stable staff
- Self-sustaining software (clean design, clean implementation, well tested with unit tests and system verification tests) => Anyone can maintain it!

The Trilinos is taking (baby) steps to address all of these issues at some level.

Sandia National Laboratories

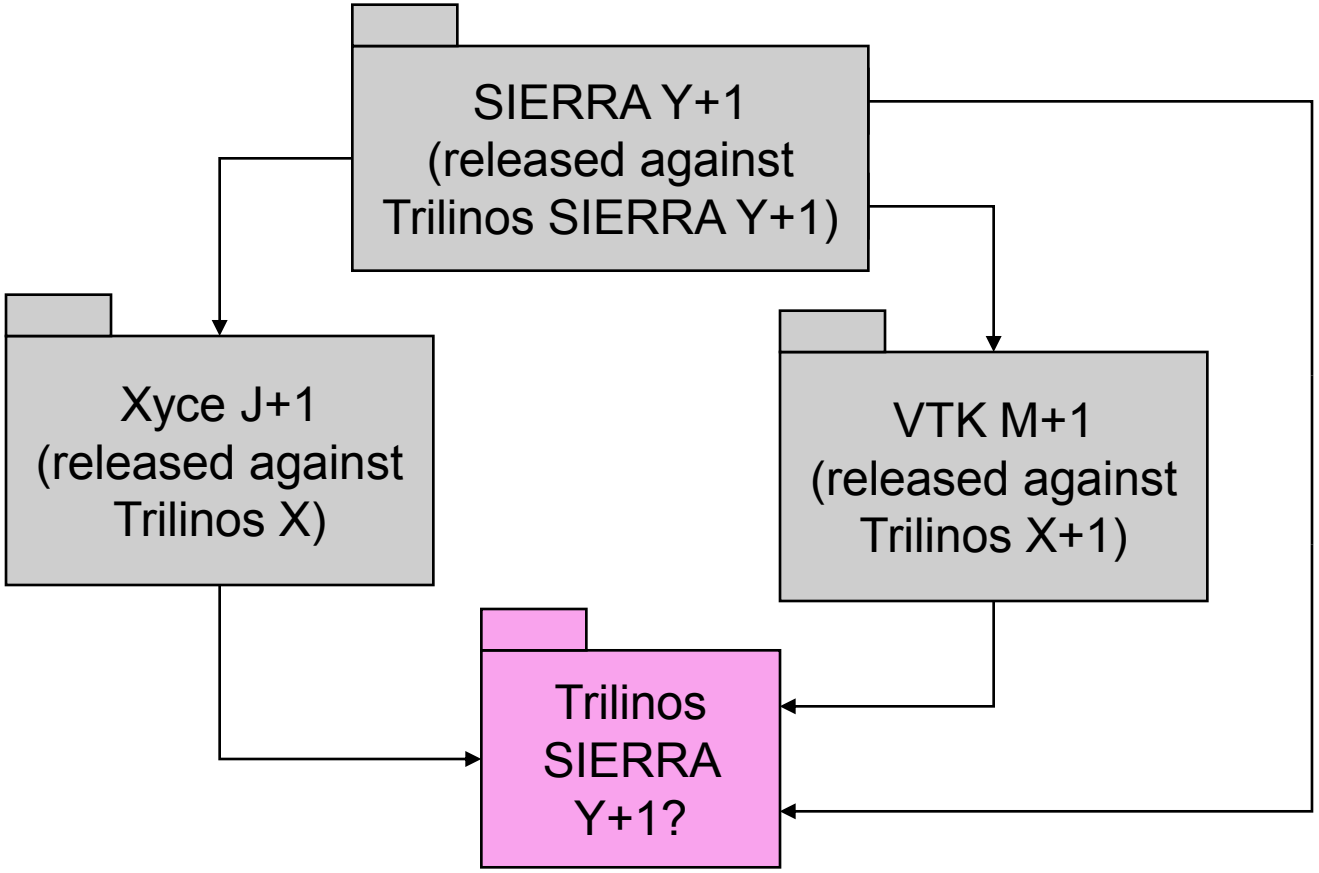
Regulated Backward Compatibility

Sandia National Laboratories

Backward Compatibility Considerations

- Backward compatibility is critical for:
  - Safe upgrades of new releases
  - Composability and compatibility of different software collections

Example of the Need for Backward Compatibility



Multiple releases of Trilinos presents a possible problem with complex applications

Solution:

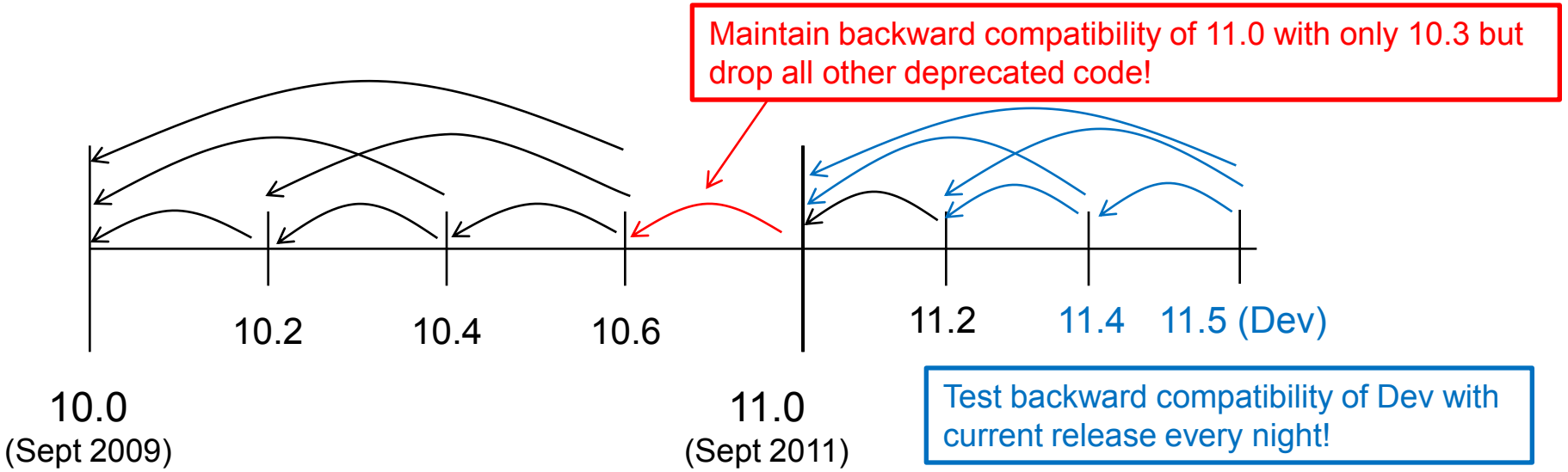
=> Provide sufficient backward compatibility of Trilinos X through Trilinos SIERRA Y+1

Backward Compatibility Considerations

- Backward compatibility is critical for:
  - Safe upgrades of new releases
  - Composability and compatibility of different software collections
- Maintaining backward compatibility for all time has downsides:
  - Testing/proving backward compatibility is expensive and costly
  - Encourages not changing (refactoring) existing interfaces etc.
    - => Leads to software "entropy" which kills a software product
- A compromise: Regulated backward compatibility (Trilinos approach)
  - Maintain a window of "sufficient" backward compatibility over major version numbers (e.g. 1-2 years)
  - Provide "Deprecated" compiler warnings
    - Example: GCC's __deprecated__ attribute enabled with –DTrilinos_SHOW_DEPRCATED_WARNINGS:BOOL=ON
  - Provide strong automated testing of Trilinos backward compatibility
  - Drop backward compatibility between major version numbers

Regulated Backward Compatibility in Trilinos

- Trilinos Version Numbering X.Y.Z:
  - X: Defines backward compatibility set of releases
  - Y: Major release (off the master branch) number in backward compatible set
  - Z: Minor releases off the release branch X.Y
  - Y and Z: Even numbers = release, odd numbers = dev
    - Makes logic with Trilinos_version.h easier
- Backward comparability between releases
  - Example: Trilinos10.6 is backward compatible with 10.0 through 10.4
  - Example: Trilinos 11.X is not compatible with Trilinos 10.Y



Example: Major Trilinos versions change every 2 years with 2 releases per year

Summary

Sandia National Laboratories

CS&E Software Engineering Challenge

- Progress in Computational Science and Engineering (CS&E) is occurring due to greater numbers of more complex algorithms and methods
  - **Discretization**: a) meshing, b) advanced discretizations, c) adaptively, …
  - **Parallelization**: a) parallel support, b) load balancing, …
  - **General numerics**: a) automatic differentiation, …
  - **Solvers**: a) linear-algebra, b) linear solvers, c) preconditioners, d) nonlinear solvers, e) time integration, …
  - **Analysis capabilities**: a) error-estimation, b) stability analysis and bifurcation, c) optimization, d) UQ, …
  - **New architectures**: a) multi-core, b) GPUs, …
  - **Visualization**
  - ...
- Each technology requires specialized PhD-level expertise
- Almost all technologies need to be integrated into single applications
- Set of algorithms/software is too large for any single organization to create
- Too large to be developed under single blanket of Continuous Integration (CI)

Software Engineering and Software Integration are key bottlenecks for CS&E to have the fullest impact!

Requirements/Challenges for Confederation of CS&E Codes

- Software quality and usability
  => Design, testing, collaborative development
- Building the software in a consistent way and linking
  => Common build approach?
- Reusability and interoperability of software components
  => Incremental Agile design, resource management, …
- Documentation, tutorials, user comprehension
  => SE education, better documentation and examples
- Critical new functionality development
  => Closer development and integration models
- Upgrading compatible versions of software
  => Frequent fixed-time releases
- Safe upgrades of software
  => Regulated backward compatibility, software quality
- Long term maintenance and support
  = > Stable organizations, stable projects, stable staff
- Self-sustaining software (clean design, well tested with unit tests and system verification tests) => Anyone can maintain it!

See Part II, Integration Issues at 10:50 AM

Possible topics for Round Table Discussion at 6:00 PM

THE END

Sandia National Laboratories